9. Manisha

Program Name: Manisha.java **Input File: manisha.dat**

Manisha is preparing for her boolean algebra exam where she will need to show mastery in her ability to write boolean equations from their equivalent digital electronics symbols. More specifically, Manisha will need to write several sums of products. A valid sum consists of one or more variables (denoted by a single uppercase letter) with a possible negation tied to an individual variable (denoted by the character '/').

While Manisha is confident in her abilities to derive those boolean equations, she is not so confident in her ability to format her answer properly for the auto-grader to mark her answer as correct. For an answer to be marked as correct, it needs to be formatted as follows:

1. Within a given product, products should be sorted…
   a. First by increasing alphabetical order.
   b. Second by the presence of a negated relation.
      1) I.e., `A` comes before `/A`
2. Between different products, products should be sorted…
   a. First by increasing length.
   b. Second by increasing alphabetical order.
   c. Third by the presence of a negated relation.
      1) I.e., `ABC` comes before `A/BC`
3. Extraneous spaces should be removed from the equation as a whole…
   a. Within a given product, there should not be any spaces between symbols.
   b. Between any two given products there should only be a single + with no spaces on either side.

Help Manisha by writing her a program that will format her answer in the format mentioned above.

Input:
The first line of input will consist of a single integer n ($1 \leq n \leq 100$) denoting the number of testcases to follow. The next n lines will each contain a single string indicating the sum of products that Manisha wishes to format properly. It is guaranteed that each product is fully simplified and therefore a given variable name will **not** appear more than once within a given product.

Output:
For each of Manisha's n requests, on their own line, print a single string which is formatted as described above.

Sample input:
```
3
FEDA
ABCD+BCDE+/CDEF+CDEF+/ABCD+A/BCD
ABCD   + BCDE +    /CD EF   + CDEF+/ABCD+   A/BCD+A+B+CD+PA
```

Sample output:
```
ADEF
ABCD+A/BCD+/ABCD+BCDE+CDEF+/CDEF
A+B+AP+CD+ABCD+A/BCD+/ABCD+BCDE+CDEF+/CDEF
```